AR-010-669

DSTO

A Serial Communication Interface for Data Acquisition Instrumentation in a Wind Tunnel

M. Spataro and S. Kent

DSTO-TR-0740

APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

A Serial Communication Interface for Data Acquisition Instrumentation in a Wind Tunnel

M. Spataro and S. Kent

Air Operations Division
Aeronautical and Maritime Research Laboratory

DSTO-TR-0740

ABSTRACT

The Low Speed Wind Tunnel (LSWT) at the Aeronautical and Maritime Research Laboratory (AMRL) has used a proprietary Bidirectional Parallel Interface (BPI) bus for data collection from instrumentation since 1989. As part of the ongoing development of the LSWT data acquisition system it was decided that a more reliable and faster communication scheme was required. This report describes the unique system of hardware and software developed to enable the VMEbus-based instrumentation modules to communicate with a Host computer over an ethernet network.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Victoria 3001 Australia

Telephone: (03) 9626 7000
Fax: (03) 9626 7999

AR-010-669
November 1998

APPROVED FOR PUBLIC RELEASE

A Serial Communication Interface for Data Acquisition Instrumentation in a Wind Tunnel

Executive Summary

The Low Speed Wind Tunnel (LSWT) at the Aeronautical and Maritime Research Laboratory (AMRL) has used a proprietary Bidirectional Parallel Interface (BPI) bus for data collection from instrumentation since 1989. As part of the ongoing development of the LSWT data acquisition system it was decided that a more reliable and faster communication scheme was required.

After cost and development times were considered, an ethernet network, using a Unix-based server as the Host computer to collect data, was selected as a replacement for the BPI. A number of the instrumentation modules in the LSWT are based on the VMEbus and have no ethernet capability. It was originally intended to connect these modules directly to the ethernet network, but this scheme depended on the availability of a VMEbus ethernet adapter that could be driven by low level software, without the need for a disk-based operating system. The development of such a system was not possible within the given time restraints, so the concept of a personal computer (PC) acting as a bridge between the ethernet and the VMEbus modules using an RS-232 serial link to each module, was developed.

The bridge PC is known as the "Serial Hub", and allows the VMEbus modules to appear to be on the network, with their own IP number and the ability to receive and transmit both narrowcast and broadcast ethernet messages.

This system has proved to be a reliable and versatile communications scheme that is capable of sustaining data transfer rates considerably higher than the original BPI bus.

Authors

Michael Spataro
Air Operations Division

Michael Spataro completed an Honors degree in Electrical and Computer Systems Engineering at Monash University in 1988. He has been employed at the Aeronautical and Maritime Research Laboratory since 1990. He has worked in the areas of electronic design, process control, and software development before becoming involved in the LSWT data acquisition upgrade project in 1996.

Steven Kent
Air Operations Division

Steven Kent is a Senior Technical Officer who has completed both a Certificate of Technology and an Associate Diploma in Electronics Engineering. He has worked for Air Operations Division for over 11 years and has gained extensive experience in data acquisition applications including hardware and software development for wind tunnel systems as well as field trial work involving Royal Australian Navy helicopters and ships. He has also been involved in the development, construction and maintenance of systems for flight simulation and operational research.

Contents

1. Introduction

The Low Speed Wind Tunnel (LSWT) at the Aeronautical and Maritime Research Laboratory (AMRL) has used a proprietary Bidirectional Parallel Interface (BPI) bus [Ref. 1] for data collection from instrumentation since 1989. As part of the ongoing development of the LSWT data acquisition system it was decided that a more reliable and faster communication scheme was needed. Several alternatives were considered [Ref. 2], including "FieldBus", GPIB (IEEE-488), various network options, and shared memory. After cost and development times were considered, an ethernet network, using a Unix-based server as the Host computer (designated Bernoulli [Ref. 3]) to collect data, was selected as a replacement for the BPI.

A number of the instrumentation modules in the LSWT are based on the VMEbus [Ref. 4]. Each module is controlled by a Motorola MC68000 microprocessor running proprietary assembly language software. It was originally intended to connect these modules directly to the ethernet network. This scheme depended on the availability of a VMEbus ethernet adapter that could be driven by low level software, without the need for a disk-based operating system. Extensive evaluation of an externally sourced VMEbus-based ethernet adapter proved that this option was not possible within reasonable time constraints. As an alternative, the concept of a personal computer (PC) (Section 2.3) acting as a bridge between the ethernet and the VMEbus modules using an RS-232 serial link to each module, was developed. The bridge PC is known as the "Serial Hub", and the unique system it uses to communicate with the VMEbus modules is described in the following section.

2. System Outline

2.1 Introduction

The instrumentation system in the LSWT communicates with the Host computer via an ethernet network. The VMEbus-based instrumentation modules, which have no ethernet capability, use a serial communications link to pass data to the Serial Hub. The Serial Hub allows the VMEbus modules to appear to be on the network, with their own IP number and the ability to receive and transmit both narrowcast and broadcast ethernet messages (Figure 1).

The communications scheme implemented allows data flow to be driven by either the VMEbus modules or the Host. Data from the modules is stored by the Serial Hub, which then transmits an ethernet packet to the Host computer when a complete "frame" has been received. Two modes are defined, one allowing the Host computer to set up module parameters and the other allowing modules to generate a stream of data which the Serial Hub sends to the Host computer.

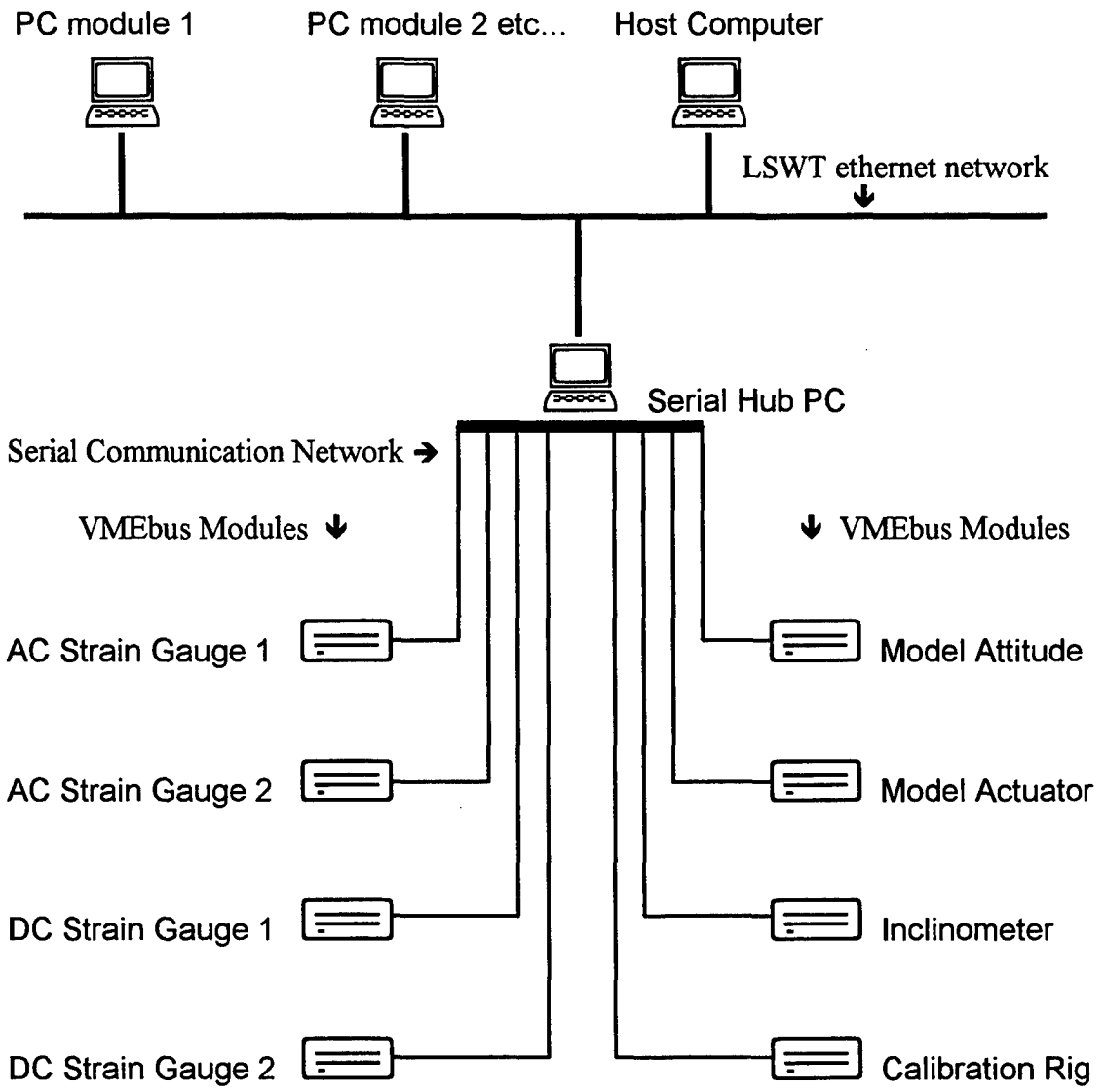

Figure 1: LSWT Ethernet and Serial networks.

Ethernet packets passed between the Serial Hub and the Host computer are in the User Datagram Protocol (UDP) format [Ref. 5]. The VMEbus modules each have an IP number assigned to them (declared in a series of .INI files). The Serial Hub accepts packets directed at these IP numbers, and converts the messages into a format that can be transmitted to the modules.

2.2 Serial Communications

The requirements of the serial interface between the Serial Hub and the VMEbus modules are as follows:

- the Host computer may send commands and setup parameters to the modules; and
- the module may send data via ethernet packets to the Host computer.

The serial interface scheme is based on packets 5 bytes in length, which are passed at 38400 baud over an RS-232 link in the following format:

byte:	1	2	3	4	5
char:	$	#	vector	dHi	dLo

The first two characters form the header. The third character, known as the 'vector', identifies the data item that follows in the next two bytes. Both the Serial Hub and the VMEbus modules send packets in this format over the serial link.

Data in the VMEbus modules are organised as 16 bit words referenced by an 8 bit vector. The valid vectors range from 0x60 (hexadecimal) to 0xFF, where the odd vectors are used for "write" operations, and the even vectors for "read" operations. Each module also reserves one vector (0x62) to allow access to an identification string, which contains the module's name and software version. This method of arranging the data, which will be referred to later as a "database", was retained from the original BPI data bus system.

2.3 Serial Hub

The Serial Hub is a PC using an Intel Pentium processor running at 133 MHz, and is fitted with a 3Com Etherlink III network card and a Hostess550 eight port RS-232 serial adapter card. The software is written in Borland C++ V4.5 and runs under DOS using On Time RTKernel V4.5 [Ref. 6]. The ethernet and database software is based on code described in Ref. 5. This code was augmented with class oriented modules to drive the serial communication ports, and compiled to produce the executable code SERHUB.EXE.

On power up, the PC will load the "packet driver" (driver for the ethernet card), before using FTP to copy up to eight setup (.INI) files from the Host computer. Following this, it will change directory to C:\LSWT\SOFTWARE\SERHUB and run SERHUB.EXE. This program will search the directory for the files copied from the Host computer, namely VMECHANx.INI, where x is a number from 1 to 8. These files contain the setup information for the channel to be established, including network information (eg. module IP number, Host IP number) and serial communication setup. The port on the Hostess550 adapter that is assigned to a VMEbus module will correspond to the number of the .INI file. A "virtual channel" will then exist for each VMEbus module, which consists of:

- an ethernet interface with its own IP number;

- a database similar in form to that of the VMEbus modules (which will mirror certain parts of the module's database); and
- a serial link to the module.

Information from ethernet packets and serial packets may be displayed on the Serial Hub screen. User commands to control screen output are single keystroke entry, and are listed on the top of the screen. This screen output is used mainly for diagnostic purposes, and is not needed during normal operation where there will be no screen or keyboard connected to the PC.

The operating system chosen to run on the Serial Hub PC is MS-DOS V6.2, and SERHUB.EXE is based on a multi-tasking software system called RTKernel (RTK). RTK was used for the following reasons:

- it is a proven real time multi-tasking software;
- the authors have previous experience with RTK;
- RTK provides support specifically for the Hostess multi-port serial I/O card; and
- to be consistent with other PC's on the LSWT ethernet network.

2.4 VMEbus Modules

The VMEbus instrumentation modules required extensive changes to both hardware and software to allow them to communicate with the Serial Hub via a serial communications link.

The BPI card was replaced with a serial communications card, which was built from a modified version of an existing AMRL design, and incorporates a Motorola MC68681 dual asynchronous receiver/transmitter (DUART) chip [Ref. 7]. The card uses interrupts on both data transmit and data receive for service requests to the CPU, and does not have any memory chips fitted. Removing the BPI card from the VMEbus modules had the following consequences for the communication scheme:

- the Host computer can no longer assert a hardware reset to the VMEbus modules, and the modules must be reset manually;
- the modules may no longer flag an error condition to the Host computer using a hardware signal, and this is replaced by an error word stored in vector 0x60; and
- the interrupt structure on the VMEbus modules is completely rearranged. The hardware interrupt vectors (data base selection vectors 0x60 to 0xFF), which were previously supplied to the VMEbus by the BPI bus, are now implemented as software vectors, with the vector being supplied in the serial packet.

Routines were added to the VMEbus module code to facilitate the transmission and reception of serial packets, allowing data write and read operations to and from the vectors. The main program loop was modified to incorporate the ability to continuously self-trigger data acquisition cycles and data streaming cycles (Section 2.5).

New vectors were added to various VMEbus modules, the common ones being:

- 0x60: error code; and
- 0x6f: streaming ON/OFF command.

Details on the hardware and software modifications to the VMEbus modules are given in Section 4, and a full listing of vector assignments is given in Appendix A.

2.5 VMEbus Module Streaming Modes

The VMEbus modules may be in one of two modes: streaming ON or streaming OFF (default). Streaming ON implies that the module generates a constant stream of serial packets, which are arranged in "frames". Each module has its own frame, which is a defined sequence of about 6 to 10 packets. Each module repeatedly sends its frame to the Serial Hub. The data in these packets is stored in the Serial Hub's database and automatically transmitted on the LSWT ethernet network when the frame is completed. The Host computer receives these messages and also stores them in a database. The data streamed by the module is that which is most important to the Host computer (i.e. a subset of the full amount of data available), and will include error and status information.

The update rate of the packets streamed from the module to the Serial Hub may be a maximum of 300 packets per second, which results in a frame rate of about 30 frames per second (i.e. 300 packets per second with a frame size of 10 packets). This will vary between modules due to differing frame sizes. The rate of ethernet packet transmission may be modified by the Host computer to be a fraction of the frame rate [Ref. 5]. The Host may send information (i.e. "direct write") to any module one word at a time, while streaming is ON or OFF. The Serial Hub decodes the ethernet packet, and a serial packet is sent to the target module. This is used for command purposes, for example to switch streaming ON or OFF, and to send initialization data to the modules.

With streaming OFF, no packets are sent by the modules unless requested, thus the hub sends no ethernet packets. Direct writes may be done, and also "direct reads" may be executed. A direct read is a four-part operation that takes data from the module to the host without being stored in the Serial Hub database. It commences with the Serial Hub decoding an ethernet message and generating a serial packet to send to the VMEbus module. The module responds with a packet and the hub sends an ethernet message containing the single piece of response data back to the Host. A variation of the direct read is the "direct string read", where the module ID string is read by a series of single reads initiated by the Serial Hub on request from the Host. When all the packets have been received from the module, an ethernet packet containing the ID string is returned to the Host. If the module does not respond to the string read request, the string returned will be similar to "Module ID string not yet read". Details of these command sequences are given in Figure 2.

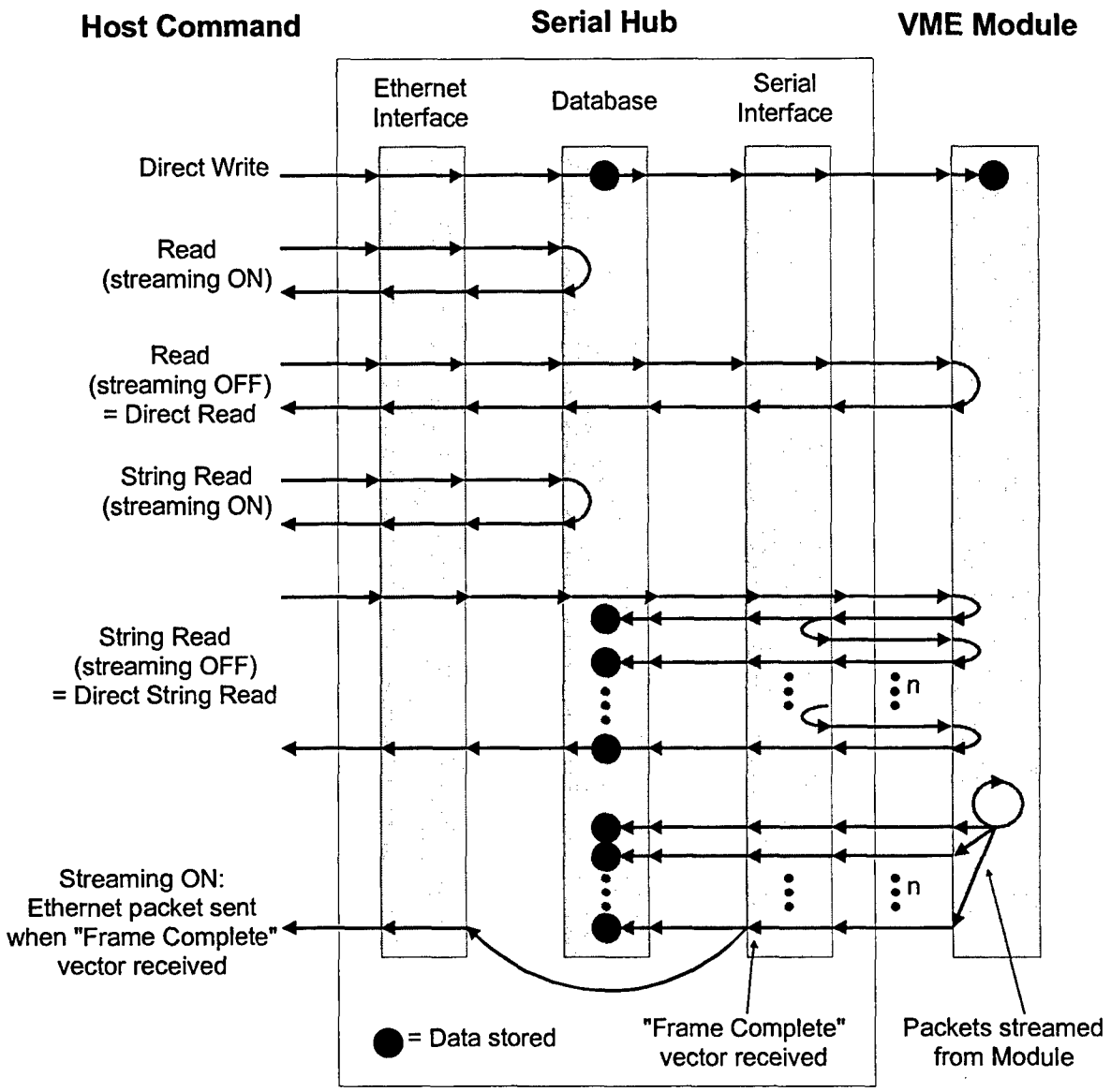

Figure 2: Data and command flow in the Serial Hub

The streaming OFF mode is best used for the setup phase where many single read and write operations are needed to initialize the VMEbus modules. The streaming ON mode is best used for the data acquisition phase. The modules are not necessarily all in the same mode, and may have streaming switched ON and OFF by sending a direct write command to the appropriate IP address.

After a reset, the default mode is streaming OFF for all the VMEbus modules. The Serial Hub keeps track of the streaming state of each module, and if the Serial Hub or any module is reset, only one stream ON or OFF command from the Host is necessary per module to re-align the two to the same mode. Programming streaming OFF as the

default streaming mode allows each module to power up in a known state to ensure the smooth operation of the Serial Hub.

2.6 Inclinometer Module

The inclinometer module is a VMEbus-based module that is a special case for two reasons:

- it requires the Serial Hub to perform calculations on the data from the module before transmitting it to the Host; and
- ethernet packets from this module are not directed at the Host, but are broadcast on the LSWT dedicated ethernet network, so that the data may be used by other PC modules.

The inclinometer module takes signals from a 3-axis accelerometer pack, which may be mounted in a wind tunnel model, from which the Pitch and Roll angles of the model are calculated. The task of executing the floating point calculations to arrive at Pitch and Roll values was shifted from the VMEbus module (running an MC68000) to the Serial Hub (running a Pentium 133 MHz with math co-processor), for increased speed and easier modification of the conversion constants in the future.

The inclinometer module streams raw accelerometer values to the Serial Hub, which performs the conversion each time a frame of packets is received. An ethernet packet that contains both the raw accelerometer values and the calculated values is then broadcast, allowing both the Host and other PC modules on the network to use the Pitch and Roll angle values.

The streaming modes for the inclinometer operate in the same way as other VMEbus modules.

3. Serial Hub Software

3.1 Introduction

The Serial Hub software is written in C++, and consists of three main classes:

- ethernet driver class;
- database class; and
- serial communication class.

Figure 2 shows the data flow between these classes.

For each of these classes, an array of nine objects is declared. Eight of these correspond to a "virtual channel" (Section 2.3), and the ninth is reserved for a "dummy" channel which allows the Serial Hub to have its own IP number and database. A virtual

channel, leading from the ethernet interface, through the database, and to the serial port, is established by a system of semaphores [Ref. 6] and function calls. Data may be passed through the virtual channel in either direction.

Files called VMECHANx.INI, where x = 1 to 8, contain the setup information for the channel to be established, both network information (eg. module IP number, Host IP number) and serial communication setup.

3.2 Serial Communications Interface [1]

The Serial Hub PC is fitted with a Hostess550 eight port RS232 serial card. Each port has its own RTKernel task that receives packets, and another task that transmits packets.

Class CSerial is defined in the file SERIAL.H and implemented in the file SERIAL.CPP. The main data items declared are:

- Port (integer) - the number of the serial port which is linked to this object;
- Baudrate (long int) – pre-defined as 38,400 baud. May be changed using #define DEFAULTBAUD;
- Trigger (integer) - when streaming mode is ON, a packet received on this serial port whose vector is equal to the Trigger (also called the "Frame Complete" vector [Figure 2]) will instigate an ethernet packet being sent to the Host;
- SerialTransmitBox (Mailbox [Ref. 5]) - signals the Serial Transmit task that a packet is to be constructed and sent; and
- SerialReplyToNetBox (Mailbox) - used by the Serial Receive task to signal the ethernet interface that a reply packet has been received from the VMEbus module. An ethernet packet will be sent as a result.

The main functions defined are:

- *InitSerialChan()* - Declare mailboxes, start tasks, and initialise serial port;
- *CloseSerialChan()* - Terminate tasks;
- *SerialTransmit()* - (not part of CSerial) Each channel starts this function as a task; and
- *SerialReceiver()* - (not part of CSerial) Each channel starts this function as a task.

3.3 Database Software

The Serial Hub implements a database for each channel that reflects the contents of the VMEbus module's database. Different types of data exchanges exist, and can be distinguished from each other in the following way:

- "Direct Write" - the Host computer writes one word to a vector in a VMEbus module. The data is stored in the Serial Hub database, and a packet containing the data is sent to the VMEbus module;

[1] An understanding of RTKernel [Ref. 6] is assumed in this section.

- "Direct Read" - the Host computer reads one word from a module. This is possible only when streaming is OFF. Data returned from the module is NOT stored in the Serial Hub database before being passed on to the Host;
- Streaming mode ON - the Serial Hub receives packets from a module that is streaming, and stores each packet in its database;
- "Read Module ID String" (only with streaming ON) - the string is read from the Serial Hub database and returned to the Host; and
- "Direct Read Module ID String" (only with streaming OFF) - the string is read one word at a time from the module, stored in the Serial Hub database, and returned to the Host.

The database code is based on the database code written for the other LSWT PC modules, and is detailed in Reference 5. Most of the functions and data items from the original code are included in the class Cdbase, which is defined in DB_HNDLR.H and DB_HNDLR.CPP. The database is initiated by calling startUpDataBaseHndlr() for each channel, which unlike the serial interface controller, creates no RTK tasks. Access to the database is achieved through one of the two following functions:

- *netAccessDataBase()* - an ethernet message from the Host computer will prompt a call to this function, which determines the type of operation required - a direct read, a direct write, or a database read of a string or a word; or
- *moduleAccessDataBase()* - each packet received by the serial interface results in a call to this function. If the packet is received while streaming is ON, the data is stored in the database. If the packet is received as a result of a direct read (i.e. streaming is OFF), then it is NOT stored.

3.4 Ethernet Interface Software

The ethernet interface for the Serial Hub is based on code written for the other LSWT PC modules, and is detailed in Reference 5. The code is essentially unchanged, with the functions *narrowcastLSWT_DataBase()* and *broadcastLSWT_DataBase()* being made part of Cdbase. The class CEnetChannel has no member functions, and only a few data members, the most important one being the struct "networkInformation". The class definitions are located in ENETDRVR.CPP and ENETCHAN.H.

The main modifications to the original ethernet code involve the ability to service multiple IP numbers. In the function *handleIP()*, the address of an incoming packet is compared with the list of IP addresses of the virtual channels, and a match returns a pointer to the relevant CEnetChannel object. This CEnetChannel pointer is passed to various functions, allowing them to access channel specific information. Functions such as *handleUDP()*, *ipReplyWithModifiedPacket()*, and the functions which they subsequently call, have a CEnetChannel object pointer passed to them.

3.5 Program Startup

The program's *main()* function, found in SERHUB.CPP, commences by calling *processIniFile()* to read the files VMECHAN1.INI through to VMECHAN9.INI. The function *getsetup()*, found in SETUP.CPP, is used to read a number of parameters at a time from each file.

Several tasks which drive screen output (such as the error monitor and the serial receiver) are started before the serial ports, database handlers, and ethernet handler are initialised. Screen output is disabled by default (for the sake of conserving CPU time), but can be enabled by an operator striking the "O" key.

The program will then wait to process packets from either the ethernet port or the serial ports, until it is terminated by an Escape or "Q" character input by an operator from the keyboard. The ethernet, serial and database handlers will then be shut down, and the screen output tasks terminated.

3.6 Inclinometer Module

As described in Section 2.6, the Inclinometer module is a special case. The Pitch and Roll angle calculations are executed by the *SerialReceiver()* task calling the function *evaluateInclinometer()* when the "Frame Complete" vector is received from the inclinometer. The calculated values of Pitch, Roll, Temperature and Error are placed in the database before an ethernet packet, containing all the raw and calculated values, is broadcast to the rest of the networked computers.

3.7 Building Serhub.exe

The Serial Hub software is written with the Borland C++ V4.5 environment running under Microsoft Windows V3.11, using the Serial Hub PC as the development platform. It is a DOS program that uses Real Time Kernel (RTK) V4.5 to provide the multitasking capability. The source code and project files reside in C:\LSWT\SOFTWARE\SERHUB. The RTK files needed are in subdirectories of C:\RTKC45.

To build the application SERHUB.EXE, run Borland C++ by clicking on the relevant icon under Windows. Select the "Project" menu and open "serhub.prj". After editing the desired source files, select Project::Build to create a new executable file. This program will not run properly under Windows, so to test the new version, exit back to the DOS prompt and type "serhub" from the directory C:\LSWT\SOFTWARE\SERHUB.

4. VMEbus Module Modifications

4.1 Introduction

To allow the VMEbus modules in the LSWT to communicate with the Serial Hub, the BPI interface card was replaced with an RS-232 serial communications card. The board chosen was an AMRL designed DUART card that required a number of changes (Section 4.2.1) to suit this application.

Significant software changes (Section 4.3) were made to implement the serial communications scheme, most of which are common to all the VMEbus modules.

4.2 Hardware Modifications

4.2.1 Modifications to DUART card

The original programmable array logic (PAL) chips (20L10) used for address and control line decoding are discontinued devices, so the current component equivalent of the 20L10 (the PALCE22V10-15PC) using generic array logic (GAL) technology was used [Ref. 8]. The GAL technology also has the advantage of being erasable and re-programmable.

The two inputs to the GAL from the DUART chip (DTACK/DUARTACK and IRQ/INT) require 4.7k (¼ watt) pull-up resistors to achieve correct voltage interfacing between the TTL outputs of the DUART chip and the CMOS inputs of the GAL.

New GAL equations[2] were required for correct interfacing with the VMEbus for the following reasons:

- to implement interrupts for faster operation (original DUART card implementation used polled access of data registers);
- memory device select functions (RAM/ROM) are not required, as the cards are not loaded with any memory chips. The existing memory cards in each module are retained, and preclude the need for extra memory; and
- critical timing requirements of the interrupt acknowledge cycle and correct operation of IACKIN and IACKOUT [Ref. 4].

4.2.2 Modifications to Module Hardware

The BPI card was removed from each module and replaced with the DUART card. As there is no longer a hardware module address [Ref. 1], each module has been assigned an IP address (Sections 2.1 and 3.4) that is decoded by the Serial Hub which then transmits and receives data through the appropriate RS-232 port. Removal of the BPI card has also removed the ability of the module to flag the host computer of error

[2] See AMRL drawing number 61183-A1 for GAL equation details.

conditions, and the ability of the host computer to remotely reset the VMEbus modules [Ref. 1], therefore the modules must be reset manually.

The Strain Gauge modules use IRQ 1 and 3 for opto-isolator and analog-to-digital conversion (ADC) interrupts respectively. IRQ 5, which was the interrupt line used for the BPI card, was chosen to be used by the DUART card.

4.3 Software Modifications

4.3.1 Interrupts

The vector system (as used by the BPI) for transmission and reception of data [Ref. 1] has been retained for ease of code transportability. There are now two vector tables - one for hardware interrupts (from ADC's, opto-isolators, and DUARTS); and one for "BPI-style" read and write vectors, which are now subroutines, not interrupt routines as in the original BPI code. These subroutines are called when a serial packet is received from the Serial Hub.

The ADC and opto-isolator interrupt vectors in the Strain Gauge modules remain the same. The DUART interrupt vector has been assigned to vector 64. This number is programmed into the DUART's interrupt vector register (IVR) in the *SetUpIRQChrPort* subroutine.

The DUART calls an interrupt when either a character has been received and is waiting in the receiver buffer first-in-first-out (FIFO) register, or when a transmitted character is transferred from the transmit holding register to the transmit shift register. When the DUART card asserts IRQ 5, an interrupt acknowledge cycle is initiated [Ref. 4] which loads interrupt vector 64. This directs the processor to execute the *duartIRQ_Handler* interrupt routine, which checks the interrupt status register (ISR) to determine if a receive or transmit interrupt occurred.

4.3.2 DUART Setup Procedure

Before the software enters the main program loop, all interrupts are disabled and Port A of the DUART is set up using the *setUpChrPort* subroutine. The receiver and transmitter are reset, as well as the Error Status and Channel A Break Change Interrupt [Ref. 4]. Mode registers 1 and 2 (MR1A and MR2A) are set up for serial communications using 8 data bits, 1 stop bit, odd parity and no request-to-send (RTS). The clock select register (CSRA) is programmed for a baud rate of 38,400. The receiver and transmitter are then enabled through the command register (CRA) and dummy reads of the status and data registers are performed to clear any false data remaining after reset.

4.4 Serial Communication Routines

Two buffers (*PutBuff* and *GetBuff*) are used to store characters to be transmitted and received (Figure 3). They are both six bytes in length. The variables *PutOff* and *GetOff* are used as offset pointers that store the position within *PutBuff* and *GetBuff* of the next character to be transmitted or received.

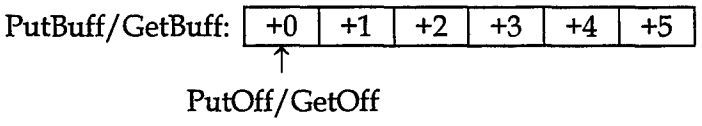

Figure 3: Transmit and Receive Buffer

As six bytes (three 16-bit words) are allocated to the buffers, and only five bytes are required for a data packet, *PutBuff*+3 and *GetBuff*+3 are skipped over when incrementing these pointers.

4.4.1 Transmit Routine

To initiate a transmit data sequence, bit 0 of the interrupt mask register (IMR) is set to 1. If the transmitter is ready, a Channel A transmitter ready (TxRDYA) interrupt will occur immediately. The interrupt is detected by the *duartIRQ_Handler* routine, which subsequently calls the *PutChar* subroutine.

The *PutChar* routine tests bit 2 of the Channel A status register (SRA) to confirm that the transmit holding register is empty, and loads *PutOff* which is initially cleared to 0, thus pointing to the first location in *PutBuff* (*PutBuff*+0). The character in this location is then written to the transmit buffer (TB). *PutOff* is then incremented to point to the next location in *PutBuff*. The *PutChar* routine is run on each transmit interrupt until the end of the data packet. If the pointer is pointing to the end of *PutBuff* (i.e. the complete data packet has been transmitted), TxRDYA is masked (turned off), *PutOff* is reset to 0 and the *packet_complete* flag is set to 1. The *packet_complete* flag is tested during the data stream routine (Section 2.5).

4.4.2 Receive Routine

A Channel A receiver ready (RxRDYA) interrupt is generated when a character has been received in Channel A and is waiting in the receiver buffer FIFO to be read by the CPU. The *duartIRQ_Handler* routine tests the ISR for this interrupt and branches to the *GetChar* subroutine. Assuming that this is the first character received the *GetOff* will be equal to 0. The character is read from the Channel A data register and *GetOff* is retrieved. As the offset is 0 the received data is compared with 0x24 ($ symbol) for correct data packet synchronisation (Section 2.2). If the data is 0x24, *GetOff* is incremented by 1 and the subroutine is exited, awaiting another receive interrupt.

Assuming a complete and correct data packet is sent to the module, the next character received is compared with 0x23 (# symbol). The third character (the vector) is checked

Assuming a complete and correct data packet is sent to the module, the next character received is compared with 0x23 (# symbol). The third character (the vector) is checked to ensure it is greater than 0x60. A vector less than 0x60 is an invalid vector and will be ignored. The fourth and fifth characters are the high and low bytes respectively of the actual data being sent and are stored in *GetBuff*+4 and *GetBuff*+5.

When the end of the data packet has been detected, the vector to jump to is retrieved from *GetBuff*+2. To point to the correct address location, the vector number is scaled and an offset is added, and the requisite vector subroutine is performed. An even vector denotes a read routine (the module must transmit data back to the Serial Hub), and an odd vector denotes the data accompanying the vector is to be stored by the module.

When a read vector is executed, the 16 bit data word to be sent to the Serial Hub is stored in *PutBuff*+4 and *PutBuff*+5, the vector number is stored in *PutBuff*+2, and the transmit interrupt is enabled to initiate the transmit data sequence. When a write vector is executed the 16 bit data word in *PutBuff*+4 and *PutBuff*+5 is stored in a memory location and no further action is necessary.

If at any stage while receiving a data packet the data does not match the correct packet sequence, *GetOff* is reset to 0 and the *GetChar* subroutine is exited. The receive routine will re-synchronise itself when the correct sequence is received. This ensures that only valid data is received.

4.5 Data Streaming

If streaming mode is ON (Section 2.5), each iteration of the main program loop will branch to the *stream* subroutine. This routine will automatically send packets containing data from a number of read vectors, which vary for each module as shown in Table 1.

Table 1: Vectors Streamed by the VMEbus Modules

Module	**Vectors streamed (in hex, in order of streaming)**[3]
6 Channel AC Strain Gauge 1 & 2 6 Channel DC Strain Gauge 1 & 2	70, 72, 74, 76, 78, 7A, 60
Strain Gauge Calibration Rig	70, 72, 74, 76, 78, 7A, 7C, 7E, 80, 82, 84, 86, 60
Inclinometer	70, 72, 74, 76, 60
Model Actuator	70, 72, 74, 76, 78, 7A, 7C, 7E, 80, 82, 84, 86, D0, D2
Model Attitude (Turntable)	C0, C2, C4, C6, C8, D0, D2, D4

The data for the first packet is retrieved and the *checkTxComplete* subroutine is run which enables the transmit interrupt and resets the *packet_complete* flag. Enabling the transmit interrupt automatically initiates *duartIRQ_Handler* and hence the *PutChar*

[3] Refer to Appendix A for definitions of the vectors.

In the case of the Model Actuator and Model Attitude modules [Refs. 9 & 10], only the data relating to those channels that are initialised and operating correctly are transmitted. In the case of the Model Actuator module, the *allowstream* variable is loaded and each bit relating to channel functionality is tested. The data for each channel is only transmitted if its corresponding bit is set to 1. In the Model Attitude module the *chan_limits* variable is tested and only the channels whose bit is set to 1 is transmitted. This speeds up transmission rate and program execution by only transmitting data for fully functioning channels.

4.6 Vector Additions And Modifications

A new vector (0x6F) that controls data streaming has been added to all modules. If a non-zero value is written to this vector then streaming is turned ON, and if zero is written then streaming is turned OFF.

For modules with ADC's (i.e. the Strain Gauges and Inclinometer), when streaming is turned ON, the modules convert data on a continuous basis, self-triggering on each iteration of the main program loop. When streaming is turned OFF, writing to the respective "convert" vectors (single channel or all channels) will manually trigger the converters.

The Inclinometer module no longer calculates the inclinometer angles, but sends the raw accelerometer data directly to the Serial Hub where it is converted to engineering units. Vectors 0x78, 0x7A, 0x7C, 0x7D, 0x7E, 0x7F, 0xAB and 0xAD, which are related to the angle calculations, but are now not relevant to the Inclinometer module, are trapped and processed by the Serial Hub (Section 2.6 and 3.6).

As the VMEbus modules can no longer flag the host computer if an error occurs, vector 0x60 (originally a vector for error strings [Ref. 1]) has been modified to contain a single 16-bit error word. This vector is streamed to the Serial Hub from all Strain Gauge modules, the Calibration Rig module and the Inclinometer module. A conversion time-out error sets bit 12 of the error word (0x1000), an over-range error sets bit 13 (0x2000) and an opto-isolator error sets bit 14 (0x4000). Any or all of these errors can occur at the same time. The error word is cleared automatically after the Host computer accesses vector 0x60.

The two modules that do not use vector 0x60 are the Model Actuator and Model Attitude modules. Instead, they store status conditions in the appropriate status vectors (see Tables A3 and A4 in Appendix A) which are streamed to the Serial Hub (see Table 1).

5. System Performance

To achieve accurate and reliable data readings during tests, a minimum data packet transmission rate of 20 ethernet packets per second from each VMEbus module with streaming ON is required. A data packet-monitoring program on the Host (ETHTEST) was used to obtain the packet rates for each module, which are shown in the following table:

Table 2: Data packet rates for the VME modules

Module	**Packet Rate (packets/second)**
6 Channel AC Strain Gauge 1 & 2 6 Channel DC Strain Gauge 1 & 2	41
Strain Gauge Calibration Rig	25
Inclinometer	52
Model Actuator	96 *
Model Attitude (Turntable)	22 - 40 †

6. Conclusion

This paper describes a unique system for interfacing VMEbus-based instrumentation modules to an ethernet network. The solution has reasonably high software development costs, but hardware costs are low, and this results in a system that is easily re-configurable for future changes. Testing in the LSWT has shown the system to be extremely reliable and capable of sustaining the data rate required.

7. Acknowledgements

The authors wish to acknowledge the assistance of Owen Holland for his work in two areas:

- the GAL programming equations to enable the DUART interrupts on the VMEbus modules; and
- the code used as the basis for the ethernet interface and database software on the Serial Hub.

* This rate was achieved for two Actuator channels operating, and the rate will obviously decrease as more channels are added (Section 4.5).

† The actual rate is determined by encoder conversion times [Ref. 10].

8. References

1 Harvey, J.F. *A Data Acquisition Parallel Bus For Wind Tunnels at ARL.*
Department of Defence, Defence Science and Technology Organisation, Aeronautical Research Laboratory, Technical Memorandum, ARL-FLIGHT-MECH-TM-412, 1989.

2 Holland, O.F. *Options for Replacing the LSWT BPI bus.*
Department of Defence, Defence Science and Technology Organisation, Aeronautical & Maritime Research Laboratory, Draft Internal Memorandum, 1997.

3 Link, Y.Y. *Bernoulli User Manual*
Department of Defence, Defence Science and Technology Organisation, Aeronautical & Maritime Research Laboratory, Draft Report, 1998.

4 Motorola Inc. *The VMEbus Specification Manual, Revision C.1.*
Motorola Inc., USA, 1985

5 Holland, O.F. (1997) *The Interface between the Host and the Instrumentation Modules in the LSWT Upgrade.*
Department of Defence, Defence Science and Technology Organisation, Aeronautical & Maritime Research Laboratory, Draft Internal Memorandum, 1997.

6 On Time Informatik GMBH *RTKernel - Real Time Multitasking Kernel for C. User's Manual Version 4.5.*
On Time Informatik GMBH, 1995.

7 Motorola Inc. *MC68681 Dual Asynchronous Receiver/Transmitter (DUART).*
Motorola Inc., USA, 1983.

8 Lattice Semiconductor Corporation *GAL Data Book.*
Lattice Semiconductor Corporation, USA, 1991.

9 Kent, S.A. *A Wind Tunnel Model Control Surface Actuator Interface.*
Department of Defence, Defence Science and Technology Organisation, Aeronautical & Maritime Research Laboratory, Technical Note, ARL-TN-13, 1993.

10 Kent, S.A. *A Computer Control Interface to Operate Turntables in the Test Section of a Wind Tunnel.*
Department of Defence, Defence Science and Technology Organisation, Aeronautical & Maritime Research Laboratory, Technical Report, DSTO-TR-0622, 1998.

Appendix A

VECTOR ASSIGNMENTS FOR VMEbus MODULES (Read and Write vectors are in hexadecimal)

TABLE A1: Strain Gauge Modules (AC1 & 2, DC1 & 2, Calibration Rig):

READ VECTOR	DATA	WRITE VECTOR	DATA
60	error code word	63	trigger ADC conversion (all channels)
62	module identification string	65	clear all status words
		69	trigger ADC conversion (all channels)
		6F	data stream control
70	read ADC 1 data	71	start ADC 1 conversion
72	read ADC 2 data	73	start ADC 2 conversion
74	read ADC 3 data	75	start ADC 3 conversion
76	read ADC 4 data	77	start ADC 4 conversion
78	read ADC 5 data	79	start ADC 5 conversion
7A	read ADC 6 data	7B	start ADC 6 conversion
7C *	read ADC 7 data	7D *	start ADC 7 conversion
7E *	read ADC 8 data	7F *	start ADC 8 conversion
80 *	read ADC 9 data	81 *	start ADC 9 conversion
82 *	read ADC 10 data	83 *	start ADC 10 conversion
84 *	read ADC 11 data	85 *	start ADC 11 conversion
86 *	read ADC 12 data	87 *	start ADC 12 conversion
D0	ADC 1 conversion status		
D2	ADC 2 conversion status		
D4	ADC 3 conversion status		
D6	ADC 4 conversion status		
D8	ADC 5 conversion status		
DA	ADC 6 conversion status		
DC *	ADC 7 conversion status		
DE *	ADC 8 conversion status		
E0 *	ADC 9 conversion status		
E2 *	ADC 10 conversion status		
E4 *	ADC 11 conversion status		
E6 *	ADC 12 conversion status		
F0	calibration relay status	F1	calibration relay control
F4	conversion buffer status	F5	clear conversion buffer

* *Calibration Rig module only*

TABLE A2: Inclinometer Module:

READ VECTOR	DATA	WRITE VECTOR	DATA
60	error code word	63	trigger ADC conversion (all channels)
62	module identification string	65	clear all status words
		69	trigger ADC conversion (all channels)
		6F	data stream control
70	read ADC 1 data (X axis)	71	start ADC 1 conversion
72	read ADC 2 data (Y axis)	73	start ADC 2 conversion
74	read ADC 3 data (Z axis)	75	start ADC 3 conversion
76	read ADC 4 data (temperature)	77	start ADC 4 conversion
78 *	read calculated Roll		
7A *	read calculated Pitch		
7C *	read required Roll offset	7D *	set required Roll offset
7E *	read required Pitch offset	7F *	set required Pitch offset
		AB *	set zero alignment
		AD *	set QFLEX transducer model
D0	ADC 1 conversion status		
D2	ADC 2 conversion status		
D4	ADC 3 conversion status		
D6	ADC 4 conversion status		
F4	conversion buffer status	F5	clear conversion buffer

* *Trapped and handled by the Serial Hub*

TABLE A3: Model Actuator Module:

READ VECTOR	DATA	WRITE VECTOR	DATA
60	error code word	65	clear all status words
62	module identification string	67	clear time-out status word
		69	trigger actuator movement
		6F	data stream control
70	read LVDT reading channel 1	71	set LVDT target channel 1
72	read LVDT reading channel 2	73	set LVDT target channel 2
74	read LVDT reading channel 3	75	set LVDT target channel 3
76	read LVDT reading channel 4	77	set LVDT target channel 4
78	read LVDT reading channel 5	79	set LVDT target channel 5
7A	read LVDT reading channel 6	7B	set LVDT target channel 6
7C	read LVDT reading channel 7	7D	set LVDT target channel 7
7E	read LVDT reading channel 8	7F	set LVDT target channel 8
80	read LVDT reading channel 9	81	set LVDT target channel 9
82	read LVDT reading channel 10	83	set LVDT target channel 10
84	read LVDT reading channel 11	85	set LVDT target channel 11
86	read LVDT reading channel 12	87	set LVDT target channel 12
88	read upper limit channel 1	89	set upper limit channel 1
8A	read upper limit channel 2	8B	set upper limit channel 2
8C	read upper limit channel 3	8D	set upper limit channel 3
8E	read upper limit channel 4	8F	set upper limit channel 4
90	read upper limit channel 5	91	set upper limit channel 5
92	read upper limit channel 6	93	set upper limit channel 6
94	read upper limit channel 7	95	set upper limit channel 7
96	read upper limit channel 8	97	set upper limit channel 8
98	read upper limit channel 9	99	set upper limit channel 9
9A	read upper limit channel 10	9B	set upper limit channel 10
9C	read upper limit channel 11	9D	set upper limit channel 11
9E	read upper limit channel 12	9F	set upper limit channel 12
A0	read lower limit channel 1	A1	set lower limit channel 1
A2	read lower limit channel 2	A3	set lower limit channel 2
A4	read lower limit channel 3	A5	set lower limit channel 3
A6	read lower limit channel 4	A7	set lower limit channel 4
A8	read lower limit channel 5	A9	set lower limit channel 5
AA	read lower limit channel 6	AB	set lower limit channel 6
AC	read lower limit channel 7	AD	set lower limit channel 7
AE	read lower limit channel 8	AF	set lower limit channel 8
B0	read lower limit channel 9	B1	set lower limit channel 9
B2	read lower limit channel 10	B3	set lower limit channel 10
B4	read lower limit channel 11	B5	set lower limit channel 11
B6	read lower limit channel 12	B7	set lower limit channel 12
D0	actuator motor status	D1	enter manual test mode
D2	actuator time-out status	D3	motor direction (manual test mode)
D4	upper limit set status	D5	exit manual test mode
D6	lower limit set status	D7	re-initialise all channels
D8	actuator functioning status	D9	power relay control
		DB	pulse mode on (manual test mode)
		DD	pulse mode off (manual test mode)

TABLE A4: Model Attitude Module:

READ VECTOR	DATA	WRITE VECTOR	DATA
60	error code word	65	clear all status words
62	module identification string	69	trigger turntable movement
		6B	stop all motors immediately
		6D	read all resolver channels
		6F	data stream control
70	read port limit test section 1 upper	71	set port limit test section 1 upper
72	read port limit test section 1 lower	73	set port limit test section 1 lower
74	read port limit test section 2 upper	75	set port limit test section 2 upper
76	read port limit test section 2 lower	77	set port limit test section 2 lower
78	read port limit balance	79	set port limit balance
80	read starboard limit test section 1 upper	81	set starboard limit test section 1 upper
82	read starboard limit test section 1 lower	83	set starboard limit test section 1 lower
84	read starboard limit test section 2 upper	85	set starboard limit test section 2 upper
86	read starboard limit test section 2 lower	87	set starboard limit test section 2 lower
88	read starboard limit balance	89	set starboard limit balance
90	read target angle test section 1 upper	91	set target angle test section 1 upper
92	read target angle test section 1 lower	93	set target angle test section 1 lower
94	read target angle test section 2 upper	95	set target angle test section 2 upper
96	read target angle test section 2 lower	97	set target angle test section 2 lower
98	read target angle balance	99	set target angle balance
		A1	allow test section 1 upper to turn
		A3	allow test section 1 lower to turn
		A5	allow test section 2 upper to turn
		A7	allow test section 2 lower to turn
		A9	allow balance to turn
		AB	do not allow any turntables to turn
		B1	synch. test section 1 upper & lower
		B3	synch. test section 2 upper & lower
		B5	synch. test section 1 upper, lower & balance
		B7	synch. test section 2 upper, lower & balance
		B9	synch. test section 1 lower & balance
		BB	synch. test section 2 lower & balance
		BD	no synchronisation
C0	read current angle test section 1 upper		
C2	read current angle test section 1 lower		
C4	read current angle test section 2 upper		
C6	read current angle test section 2 lower		
C8	read current angle balance		
D0	turntable movement status		
D2	channel OK & limits status		
D4	MCU & power status		

APPENDIX B

PROGRAMMING INFORMATION FOR MODIFIED EPROMs AND DUART GALs

EPROMs and GALs are programmed using the PC with the Hi-Lo Systems ALL-03A Universal Programmer connected to it. The programming software is located in the **C:\UNI-PROG** directory.

To program the EPROMs or GALs the relevant software must be executed, as shown in Table B1. Ensure that the correct EPROM or GAL manufacturer and type are selected by using the ***M*** (manufacturer) and ***T*** (type) options in the Main Menu, and load the relevant compiled code file (see Table B1) into the programmer buffer by using option 2 (Load) in the Main Menu. If programming EPROMs, select ***M*** to load Motorola S HEX format and accept the default file start address (00000000). Unused bytes can be selected as "don't care".

Select the ***P*** option to program the devices. If programming EPROMs, type ***O*** to program the odd (upper) bytes then ***E*** to program the even (lower) bytes.

The positions for odd and even EPROMs on the memory card are shown in Figure B1.

Table B1: EPROM and GAL Programming Information

Module	GALs/EPROMs	Source Code	Compiled Code	Program with
DUART Card	2 x PALCE22V10-15PC (GAL)	IC5.TDL IC6.TDL	IC5.JED IC6.JED	GAL2.EXE
AC Strain Gauge 1 AC Strain Gauge 2 AC Strain Gauge 3*	2 x 2732 (EPROM)	SGMAC6_1.S SGMAC6_2.S SGMAC6_3.S	SGMAC6_1.HEX SGMAC6_2.HEX SGMAC6_3.HEX	EPP512.EXE
DC Strain Gauge 1 DC Strain Gauge 2	2 x 2732	SGMDC6.S SGMDC6_2.S	SGMDC6.HEX SGMDC6_2.HEX	EPP512.EXE
Inclinometer	2 x 2732	INCLINO.S	INCLINO.HEX	EPP512.EXE
Calibration Rig	2 x 2732	CAL_RIG.S	CAL_RIG.HEX	EPP512.EXE
Model Actuator	2 x 2732	FA18DUAR.AS ACTUATOR.C	FA18DUAR.HEX	EPP512.EXE
Model Attitude	2 x 2732	MODELATA.AS MODELATC.C	MODELATT.HEX	EPP512.EXE

** ex-Transonic Wind Tunnel strain gauge module (used for spare parts)*

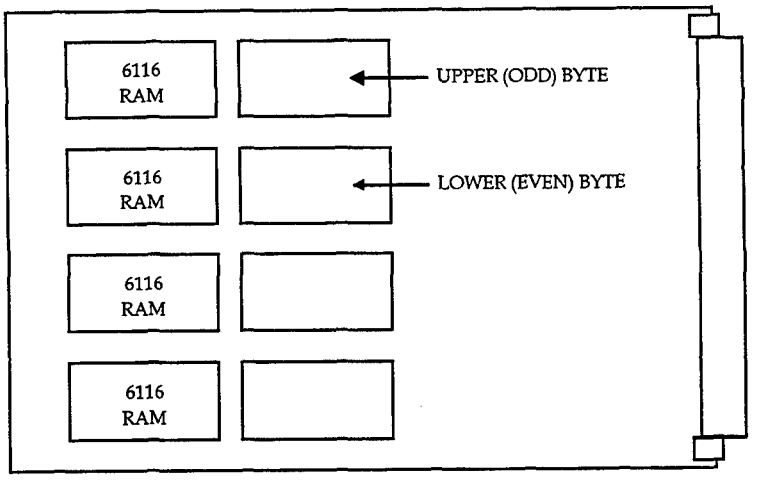

Figure B1: Memory Card EPROM Configuration

DISTRIBUTION LIST

A Serial Communication Interface for Data Acquisition Instrumentation in a Wind Tunnel

M. Spataro, S. Kent

AUSTRALIA

DEFENCE ORGANISATION

S&T Program

Chief Defence Scientist
FAS Science Policy
AS Science Corporate Management
} shared copy

Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Director General Scientific Advisers and Trials/Scientific Adviser Policy and Command (shared copy)
Navy Scientific Adviser (Doc Data Sheet and distribution list only)
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser

Aeronautical and Maritime Research Laboratory

Director

Chief of Air Operations Division
N. Pollock
N. Matheson (2 copies)
Y. Link (3 copies)
O. Holland
P. Malone
J. Clayton
D. Carnell
I. Amott
L. Erm
H. Quick
G. Ainger
A. Gonzales
S. Lam
C. Edwards
R. Toffoletto
M. Glaister
Authors: M. Spataro (2 copies)
S. Kent (2 copies)

DSTO Library

Library Fishermens Bend
Library Maribyrnong
Library Salisbury
Australian Archives
Library, MOD, Pyrmont (Doc Data sheet only)

Capability Development Division

Director General Maritime Development (Doc Data Sheet only)
Director General Land Development (Doc Data Sheet only)
Director General C3I Development (Doc Data Sheet only)

Corporate Support Program (libraries)

OIC TRS, Defence Regional Library, Canberra
Officer in Charge, Document Exchange Centre (DEC), (Doc Data Sheet and distribution list only)
*US Defence Technical Information Center, 2 copies
*UK Defence Research Information Centre, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Deakin University, Serials Section (M list), Deakin University Library, Geelong
Senior Librarian, Hargrave Library, Monash University
Librarian, Melbourne University

OTHER ORGANISATIONS

NASA (Canberra)
AGPS

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS

INSPEC: Acquisitions Section Institution of Electrical Engineers
Engineering Societies Library, US
Documents Librarian, The Center for Research Libraries, US

INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US

SPARES (10 copies)

Total number of copies: 64

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

A Serial Communication Interface for Data Acquisition Instrumentation in a Wind Tunnel

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

M. Spataro and S. Kent

5. CORPORATE AUTHOR

Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Vic 3001 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-0740	AR-010-669	Technical Report	November 1998

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
M1/9/521	98/179	DSTO	30	10

13. DOWNGRADING/DELIMITING INSTRUCTIONS

Not Applicable

14. RELEASE AUTHORITY

Chief, Air Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

low speed wind tunnels, data communicating systems, VMEbus (computer bus), ethernet (local area network system), computer interfaces

19. ABSTRACT

The Low Speed Wind Tunnel (LSWT) at the Aeronautical and Maritime Research Laboratory (AMRL) has used a proprietary Bidirectional Parallel Interface (BPI) bus for data collection from instrumentation since 1989. As part of the ongoing development of the LSWT data acquisition system it was decided that a more reliable and faster communication scheme was required. This report describes the unique system of hardware and software developed to enable the VMEbus-based instrumentation modules to communicate with a Host computer over an ethernet network.